Test Results of Autonomous Behaviors for Urban Environment Exploration

G. Ahuja, D. Fellars, G. Kogut, E. Pacis Rius*, B. Sights, H.R. Everett
Space and Naval Warfare Systems Center, Pacific
53560 Hull Street, San Diego, CA 92152
*estrellina.pacis@navy.mil

ABSTRACT

Under various collaborative efforts with other government labs, private industry, and academia, SPAWAR Systems Center Pacific (SSC Pacific) is developing and testing advanced autonomous behaviors for navigation, mapping, and exploration in various indoor and outdoor settings. As part of the Urban Environment Exploration project, SSC Pacific is maturing those technologies and sensor payload configurations that enable man-portable robots to effectively operate within the challenging conditions of urban environments. For example, additional means to augment GPS is needed when operating in and around urban structures. A MOUT site at Camp Pendleton was selected as the test bed because of its variety in building characteristics, paved/unpaved roads, and rough terrain. Metrics are collected based on the overall system's ability to explore different coverage areas, as well as the performance of the individual component behaviors such as localization and mapping. The behaviors have been developed to be portable and independent of one another, and have been integrated under a generic behavior architecture called the Autonomous Capability Suite. This paper describes the tested behaviors, sensors, and behavior architecture, the variables of the test environment, and the performance results collected so far.

Keywords: robotics, urban environment, performance testing, exploration, mapping, localization

I. MOTIVATION

It is projected that future conflicts will increasingly occur in urban settings, and such settings pose unique challenges to effective operation of unmanned ground vehicles. Given the established DoD policy for increased use of unmanned systems in military operations, the fundamental purpose of SSC Pacific's Urban Environment Exploration (UrbEE) effort is to enhance existing teleoperated systems with autonomous behaviors that can reliably perform in challenging urban environments. For example, urban structures can inhibit reliable GPS signals due to satellite occlusion and multi-path interference. As a consequence, GPS alone is not sufficient to continuously determine vehicle position and direct vehicle motion in urban settings. For a mobile robot to consistently perform in an urban environment, it must be equipped with additional means to augment GPS when operating in and around congested areas. Urban settings also pose unique navigation impediments (e.g., obstacles and varying terrain) that the robot must be able to effectively perceive and accommodate.

The objective of the 4-year UrbEE effort is to test various component behaviors out of the laboratory environment against realistic urban environment variables in order to report the readiness of autonomy for use in urban operations. Our goal is to produce a reliable system that will allow a small UGV to explore and map an urban area consisting of multiple building structures (including multi-stories) and varying terrain (paved/unpaved roads, ditches, vegetation, etc.). Performance tests are being conducted on an ongoing basis to characterize the behaviors with appropriate metrics and identify their maturity levels. Some of the performance questions that are being addressed include:

- How precisely can a robot's position be calculated in the presence of GPS vs. GPS-denied areas?
- How precisely can a robot map inside buildings with varying clutter?
- What is the accuracy of the robot's execution of a given path with and without the use of GPS?
- What is the maximum size of the operating environment in which the behaviors can effectively operate?

This paper reports the findings from our first year of UrbEE testing in FY08. Descriptions of our test set-up, the behavior suite, and test results follow in the later sections.

II. TEST SET-UP

2.1 Environment

A MOUT site at Camp Pendleton has been chosen as our performance testing location; the same site is used for military training. The range consists of residential sections, including a gas station, houses, apartment buildings, a school, a playground, a business district consisting of a hotel, office buildings, and a town square. There are a total of 29 buildings (one, two- and three-story), 14 that are intact and 15 that have been partially damaged (see Figure 1). In addition, there are nine ghost buildings to represent buildings that have been completely destroyed. Testing in FY08 was scheduled every month and has been solely on paved roads and in cluttered, single story buildings. FY09 tests will include non-paved roads and multiple single-story building structures; FY10 and FY11 tests will include multi-story buildings within larger exploration areas. Subsequently, we will be expanding our behavior suite yearly to meet the increasing environment challenges.

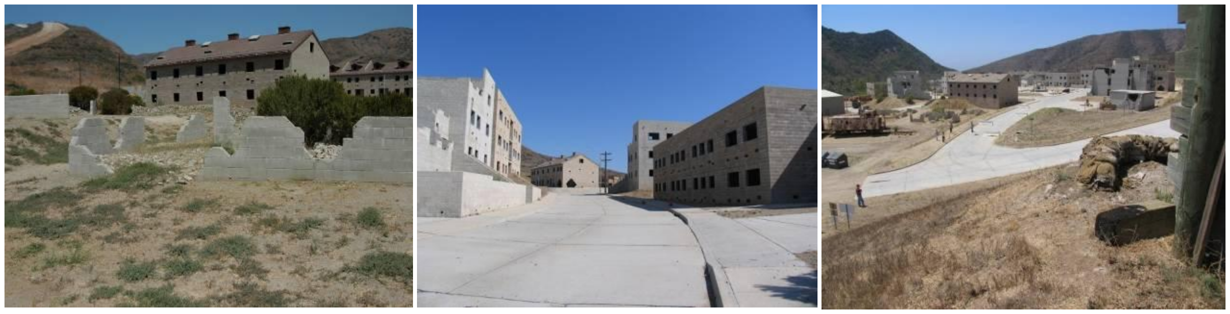


Figure 1. MOUT test site at Camp Pendleton consists of multiple buildings and varying terrain.

2.2 Platform

The platform used during FY08 testing was an iRobot Packbot Scout chassis with a first generation iRobot Navigator Payload developed for SSC Pacific under the Center for Commercialization of Advanced Technology (CCAT) program (figure 2 left). The first generation of the navigation payload contained a sensor suite that includes a 360 degree Sick ladar, a KVH gyro, a MicroStrain IMU, and a Ublox GPS. The gyro, IMU, and GPS sensors are used for navigation and localization while the ladar is used for obstacle avoidance and SLAM. In addition, the payload contains a main computer and a serial radio. The second generation payload (figure 2 right) will be used during FY09 testing. Collaboration with Foster Miller under a Cooperative Research and Development Agreement is also ongoing to integrate a similar navigation payload onto a TALON that will support the UrbEE behaviors for FY09 tests as well.

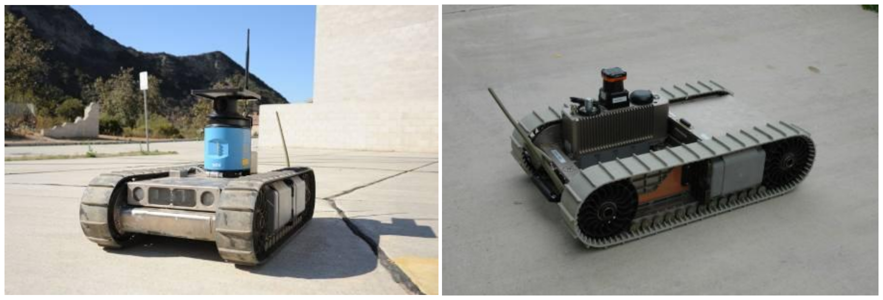


Figure 2. First generation of iRobot's Navigator Payload used for UrbEE testing in FY08 (left); 2nd generation planned for use in FY09 testing (right).

2.3 Architecture

The end-goal of this project is to provide reliable autonomous behaviors that can support a wide range of urban operations. The behaviors have been developed and integrated to be portable and independent of one another so that any subset of capabilities can be easily applied to any emergent mission or requirement. In order to facilitate this, the Autonomous Capabilities Suite (ACS) was developed by SSC Pacific. ACS is a modular software architecture supporting development and maturation of new payloads, devices, perceptions, behaviors, human-robot interaction techniques, and communication protocols on unmanned systems. Our goal is to make it easier to not only develop across any sensor or unmanned systems platform, but also easy to incorporate and test new perceptive and behavioral algorithms, as well as support any communications protocols required, without having to re-write any software that the new modules may affect/influence. Figure 3 illustrates the component structure of ACS, where devices are sensors or payloads, perceptions are derived data based on device or other perception data, behaviors are robot actions determined by the perception, device data, and/or operator input, and tasks are sequences of behaviors. The desired modularity has been achieved through standard data and command messaging between modules and use of a hierarchical state machine. The standard messaging makes it easier to add new modules, and the hierarchical state machine makes it easy to create new tasks, or sequences of behaviors, which provide new robot capabilities. The user can adjust which modules are used, the parameters the modules use to process their algorithms, and how data from each module is connected to other modules.

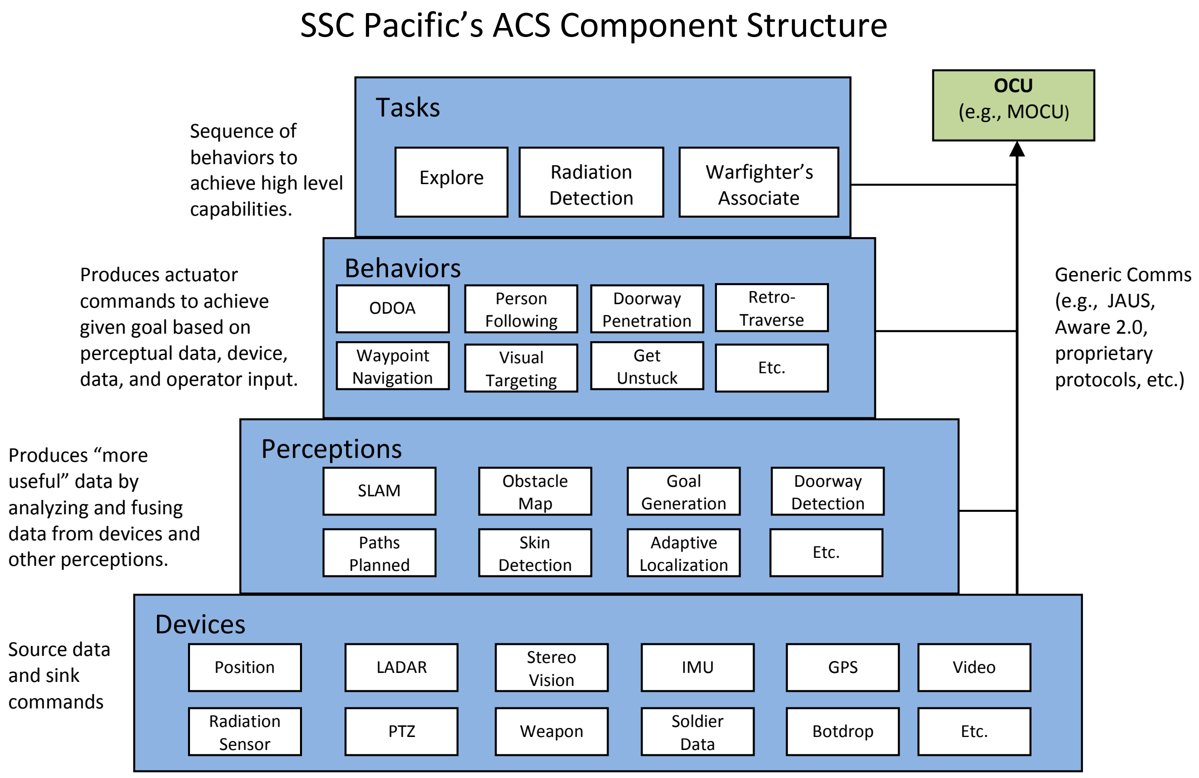

SSC Pacific's ACS Component Structure
Tasks
Sequence of behaviors to achieve high level capabilities.
Explore
Radiation Detection
Warfighter's Associate
OCU (e.g., MOCU)
Behaviors
Produces actuator commands to achieve given goal based on perceptual data, device, data, and operator input.
ODOA
Person Following
Doorway Penetration
Retro-Traverse
Waypoint Navigation
Visual Targeting
Get Unstuck
Etc.
Generic Comms (e.g., JAUS, Aware 2.0, proprietary protocols, etc.)
Perceptions
Produces "more useful" data by analyzing and fusing data from devices and other perceptions.
SLAM
Obstacle Map
Goal Generation
Doorway Detection
Paths Planned
Skin Detection
Adaptive Localization
Etc.
Devices
Source data and sink commands
Position
LADAR
Stereo Vision
IMU
GPS
Video
Radiation Sensor
PTZ
Weapon
Soldier Data
Botdrop
Etc.

Figure 3. The ACS is a modular robotic software suite that is capable of integrating advanced perceptive algorithms and behaviors on a variety of platforms because of its modularity and strong messaging backbone.

2.4 Operator Interface

ACS is designed to be independent of any particular operator control unit, and communicates with outside control software through a module called the communications module. The primary communication module used during the FY08 Pendleton tests was a Joint Architecture for Unmanned Sytems (JAUS) module, which allows communication with any JAUS-capable OCU. All our FY08 Pendleton tests used the SSC Pacific developed Multi-

Robot Operator Control Unit (MOCU) using the JAUS protocol. MOCU was developed to be easily configurable for any robot platform, user interface requirements, and mission operation. Our version of a MOCU interface to support UrbEE testing was designed for an engineer's ease of use for commanding the robot for performance testing, such as basic robot state and SLAM map information (figure 4). Some customizations were added to show more sensor information based on the behavior being executed. For example, during an exploration test run, the user-defined exploration area (yellow box), the goal points being considered by the robot (red circles), the path planned to the next chosen goal (green path), and the SLAM map are displayed. An interface design based on human factors practices were beyond the scope of the UrbEE project objectives; however, there is a separate SSC Pacific effort to develop a MOCU interface centered on human factors studies based on EOD operations. It is our hope to utilize the lessons learned to improve our testing interface.

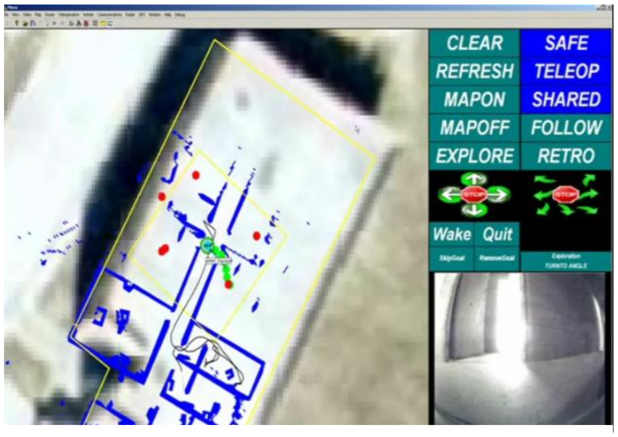


Figure 4. Screen shot of the MOCU interface used for UrbEE testing.

2.5 Behaviors

As a first step to test traditional lab solutions in near-operational environments, the first year of UrbEE testing focused on cluttered, single-story buildings and paved roads. An operator selected a building of interest using MOCU, which sent the robot to autonomously approach, enter, explore, and map the building. The core component behaviors contributing to the Exploration scenario tested are described here. Many other inherent behaviors are also used, such as obstacle avoidance, and are described in detail in [2]. Behaviors planned to be tested in the following years will be described in Section 3.0.

2.5.1 Simultaneous Localization and Mapping

Simultaneous localization and mapping (SLAM) is a method of building a map while staying referenced inside the map. The Karto mapping libraries[3] developed by SRI International were used for our tests. Karto provides an accurate pose while building the map through laser scan matching and loop closures. The map is built without any prior knowledge of the building and is useful for finding unexplored areas of the building and to plan paths within the explored areas. Staying localized within the map is crucial to assure an accurate map, which is important for finding exploration goals and path planning. The Karto mapper works well on even surfaces as long as the pose input does not have huge jumps in yaw. The stock iRobot Packbot provides poor odometry data with yaw jumps as large as 90 degrees at times when the vehicle has barely turned a few degrees. The data gets even worse on carpeted surfaces where the tracks barely move, but the odometry data keeps updating. The Karto mapper can overcome most of the poor odometry data by aligning the laser scans correctly, but the map does break sometimes. To reduce errors, the ACS mapping module provides position data from an Adaptive Extended Kalman Filter module (further described in section 2.5.4) to the Karto mapper which helps produce much better maps. Another reason for poor maps can be caused by reading scans while traversing uneven surfaces. To mitigate errors caused by poor scans, the ACS mapping module intelligently picks which scans should be added to the Karto mapper by comparing laser and orientation data timestamps and by discarding scans that were read when the pitch or roll had jumped over a certain threshold.

2.5.2 Exploration

The ACS exploration task module is a combination of position control behavior modules, including path following, goto, turnto, and get-unstuck, which are sequenced to enable robotic search of a building. Goal locations that indicate needed areas to search are determined by analyzing the SLAM output; selection of the goals influence the robot's search pattern and is determined by a method that analyzes the paths to those goals generated by the path planning perception module.

2.5.2.1 Goal Generation

The first step in automatically exploring a building of interest is to analyze the map that the robot has generated and locate the areas of the map that still need to be explored. In order to do this, SSC Pacific has developed a simple method of determining the regions between open space and unexplored space, also known as "frontiers," and grouping them into goals. Our goal-generation method is as follows:

1. For all new cells in the map, or cells in the map that have changed, count the number of cells around that cell that are empty, filled, and unknown. If the cell itself is empty or unknown, and there are at least two empty and two unknown cells around, and no more then one filled cell, then the cell is a "frontier". The "frontier" cells are stored through each addition to the map if their empty/filled/unknown status did not change, or if their neighbors' empty/filled/unknown status did not change.
2. Group all frontier cells that are within a specified distance of each other, usually a couple of cells width, into polygons. Merge all polygons that are within a specified distance of each other, usually 0.5 meters.
3. Analyze each polygon for area, width, and distance from already explored and tried goals, included inside the user-defined "exploration" region, and excluded from any user-defined "ignore regions." Retain the polygons that meet the requirements and publish them as the possible exploration goals.

2.5.2.2 Goal Selection

Analyzing a map to find exploration goals is relatively straight forward, but deciding which goal is the best one to explore at a given time can be more difficult. Also, determining when it is not actually possible to reach a given goal is important to ensure the exploration mission will complete in a timely fashion. An analysis of several different methods to select goals that have been tested; the algorithm we have developed based on our testing experience and our method for determining when a goal is not actually reachable are described below.

a. **Closest Goal.** The closest-goal selection method simply chooses the goal that is the absolute minimum distance from the robot, based on the robot and goal locations in x/y coordinates. This selection method is fast because it does not require the generation of a path; however, this means that it does not calculate the distance the robot must actually drive to reach the goal, which can result in choosing goals that are not ideal. For instance, if the entrances to two rooms are close together, the robot may find explore goals in both entrances and enter one of the rooms (figure 4 left). When it enters the room, the goal it was moving towards no longer exists, and new goals in that room are found. The new goals are most likely at the edges of the room the robot enterred. The goal in the other entryway still exists and is now closer then the goals in the current room. This scenario causes it to leave the current room and go into the other room (figure 4 right).

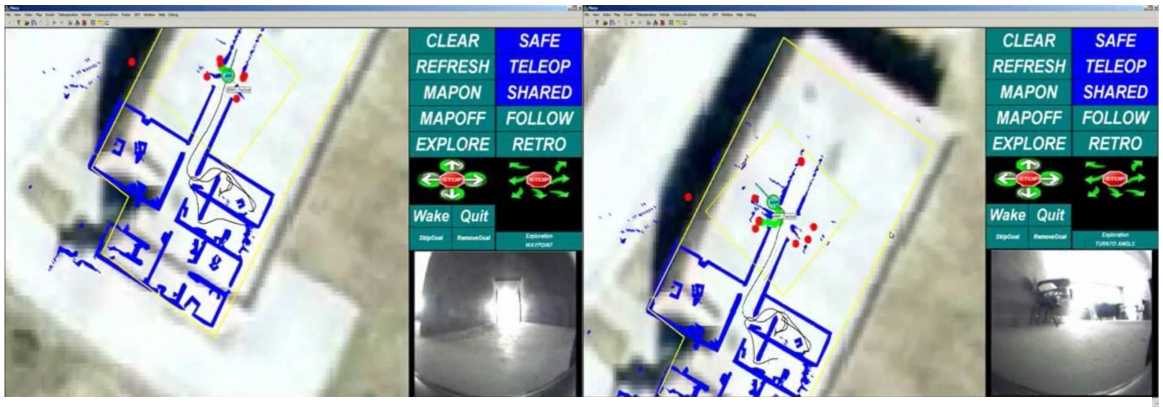


Figure 4: (Left) The robot finds goals in two different room entrances and chooses the one on the left. After reaching the first goal, the robot selects a goal in the other room because it is closer (Right). This leads to degradation of exploration performance.

b. Shortest Path: Unlike the "Closest-Goal" method, the "Shortest-Path" method plans a path to each possible goal, and then chooses the one with the shortest length. One of the problems with this method is that it does not have map data to plan the paths against, so the shortest path may not necessarily be accurate. Another problem with this method is that it can take a long time to plan paths to all the possible goals if there are a lot of goals or if the map is large. Our tests also showed that this method also does not always eliminate the problem with moving room to room without completing exploration of the rooms; therefore, more constraints, such as the amount of turning required to follow the path, need to be taken into account.

c. Farthest Goal/Longest Path: The "Farthest-Goal" and "LongestPath" methods simply choose the farthest goal from the robot, either by a simple distance calculation or by first planning a path to each goal and calculating the length of the path. These methods were originally implemented to combat a perceived tendency for the robot to stay too long in one area and not explore and map in a timely fashion. Our tests showed, however, that these methods often resulted in the robot driving back and forth across the building and never completing any of the rooms. The "Longest Path" method seemed to have the worst results since it not only caused the robot to traverse the same areas repeatedly, but also took a long time to generate paths as the map became larger, thus making it the slowest exploration method of all.

SPAWAR Derived Method: After testing for several months, it was obvious that the robot needed a more efficient way to determine the best goal to explore. The first thing we did was reduce the number of options when using the shortest path method by only planning paths to the closest (straight line) goals. We then developed rules to determine the best goal to drive to next:

1. How short is the path to the robot, with a shorter path being more desirable?
2. What is the absolute summation of heading change required to follow a path, with a smaller heading change being more desirable? In other words, the less a robot has to turn along a path, the better the goal is.
3. How much must the robot turn to follow the first waypoint on the path, with less initial turn being more desirable? This rule is meant to keep the robot from wasting time turning around unnecessarily, and also to eliminate situations where the robot enters a room and then chooses the goal in the next room over (since the robot will need to turn around to exit the room and go to the other one).

These rules have been formulated into fuzzy associative memory (FAM) rules, and degree-of-membership (DOM) functions have been developed to convert the crisp numbers into natural language for use by the fuzzy rules. The parameters for each goal are analyzed using the FAM rules to determine the desirability of choosing the goal, and the goal with the highest ranking is selected.

Deciding When to Stop Attempting to Reach a Goal: One of the problems that was discovered when initially developing and testing our exploration algorithms was the tendency for goals to be generated that cannot physically be reached by the robot, or are unreachable given the user-defined exploration zone. It is often impossible to tell that a goal is unreachable until the map has been sufficiently built around it. Therefore, we also developed rules to understand if a robot has adequately attempted to reach a goal that subsequently should be removed from consideration. The rules to remove a goal are as follows:

1. If the robot is not getting physically closer to the goal after a certain period of time.
2. If the robot is not getting closer to the goal along a path after a certain period of time.
3. If the amount of time spent trying to reach a goal is greater than it would have taken to drive straight to a point which was n times farther away than the goal was when originally chosen by the robot.

2.5.3 Path Planning

Paths to goals are calculated using SRI Karto's gradient path planning libraries[3]. The Karto path planner is a path-generation module that computes a path from point A to B using a map grid. The path planner takes a Karto map grid, a goal location, and a starting position as inputs and returns a list of path points from the starting position to the goal using a gradient descent method. Our tests show that it works reliably and efficiently but does take a considerable amount of time to compute paths as the map grows.

SSC Pacific developed an ACS PathPlanning module that wraps the Karto path planner and adds more utility. For example, as a map grows, the ACS PathPlanning module continuously updates the map grid to dynamically generate an updated path to a goal; this is accomplished by calling the Karto path planner with the new grid. The ACS Path Planning module also adds the capability to plan paths to a goal that is outside the existing map grid by extending the grid to include the goal. This is useful because as the exploration algorithm merges frontier cells into single goals, occasionally the goals' centers are outside of the existing map grid. Even if the goal were only a couple of cells outside of the Karto grid, it would not allow for generation of a path.

Another utility provided by the ACS path-planning module is the ability to compute paths to nearby points when a path to the target goal isn't returned by the Karto path planner. This is done by looking for empty cells around the goal within a specified goal radius. The ACS path-planning module adds a few other simple but very useful and effective enhancements, such as discarding paths that would send the robot outside the user-defined explore zone.

2.5.4 Adaptive Localization

An ACS adaptive localization module[4] based on an Adaptive Extended Kalman Filter (AEKF) was developed at SSC Pacific to determine the best pose from various sensor inputs in order to adapt to the operating environment the robot is in at any given time. Specifically, this helped resolve localization in the presence of intermittent GPS, and provided an overall capability for the robot to seamlessly transition between outdoor and indoor environments without requiring the operator to command the robot to change the behavior or sensors it should use.

The ACS adaptive localization module utilizes all the sensors onboard the iRobot Navigator Payload to input the following data to the AEKF:

- Odometry data for velocity and yaw-speed
- IMU data from a Microstrain 3DM-GX1 for pitch, roll, yaw, pitch-speed, roll-speed, and yaw-speed
- Fiber optic gyro data from a KVH DSP-3000 for yaw rate
- SLAM data from SRI's Karto for x, y, and yaw
- GPS data from a Ublox GPS unit for latitude, longitude (x, y), velocity, and yaw

However, during our tests, we wanted to test and demonstrate that the AEKF was capable of enhancing systems that did not have ladar or SLAM algorithms, such as most fielded EOD systems, so only the following sensors were used during the FY08 tests:

- Odometry velocity measurements
- IMU pitch and roll measurements
- Gyro yaw speed measurements
- GPS latitude and longitude (x, y), velocity, and yaw when available outdoors. (The GPS unit obviously works indoors, where most of the test data reported here was taken.)

The test scenarios included outdoor and indoor navigation, but most test results reflected indoor operation as exploration of the building was the main test objective. Though these tests were conducted mostly on flat terrain, localization was still challenging because the robot made a lot of turns, 35845 degrees (~100 full rotations) on average for the school house runs, and the robot often ran over objects on the floor that it could not detect with the ladar.

2.5.5 Get Unstuck

The presence of varying clutter was an environment variable required when we identified our test sites in order to test against realistic operating conditions. When trying to reach a goal in a cluttered room, the robot usually reaches locations where it could no longer move towards the goal as desired and would need to maneuver its way out of the tight situation it was in. In these instances, the ACS Get Unstuck behavior module backs the robot out of the area until there is enough room to turn to an open direction. This behavior works well as long as the turning radius along the path to back out of is not too small. In cases where the robot needs to make a sharp turn to avoid an obstruction, the behavior fails. Operator intervention is then needed to take control of the robot, which could be a problem in situations where communications back to the robot for tele-operation was poor. Following FY08 testing at Pendleton, new features have been added to this behavior to accommodate varying levels of clutter and turning radius.

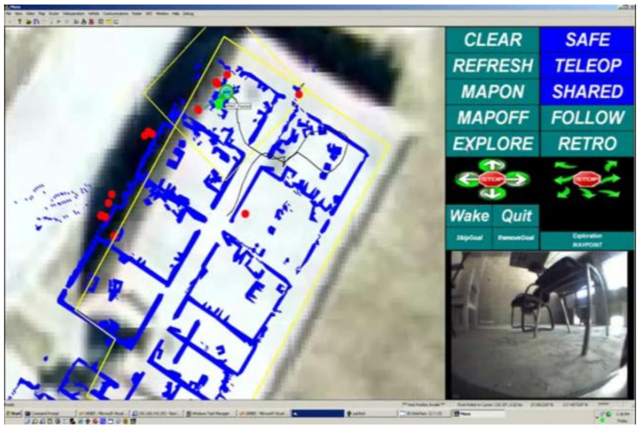


Figure 5: Bottom right window displays the camera view from the PackBot, showing an example of the amount of clutter (here, tables and desks) present.

III. TEST RESULTS

3.1 Mapping & Exploration

Analyzing the SLAM data from three complete runs at Camp Pendleton showed that we were able to achieve both accuracy and consistency in mapping. The resulting maps had an average of 91% topological correctness and 95% dimension accuracy relative to a hand-measured reference. It is important to note that the median dimension error was about 1%, meaning that most of the error occurred due to large magnitude errors in a few failure conditions. These failure conditions are discussed below and are likely not fixable without the use of 3D information. More data in a wide variety of environments is needed to demonstrate that the system will perform comparably in a range of indoor settings.

3.1.1 Ground Truth

Figure 6 is the ground truth map created from hand-measured dimensions of each room in the school house the robot was tasked to explore during the FY08 tests. From this map, 14 areas were defined as rooms by human interpretation. During two of the three runs, the stairwell in the lower right hand of the image was identified as a room, but we did not include this as a false positive in the calculations since the data available to the robot was not sufficient to detect stairwells. Future work will include the use of visual data to help identify stairways and other objects.

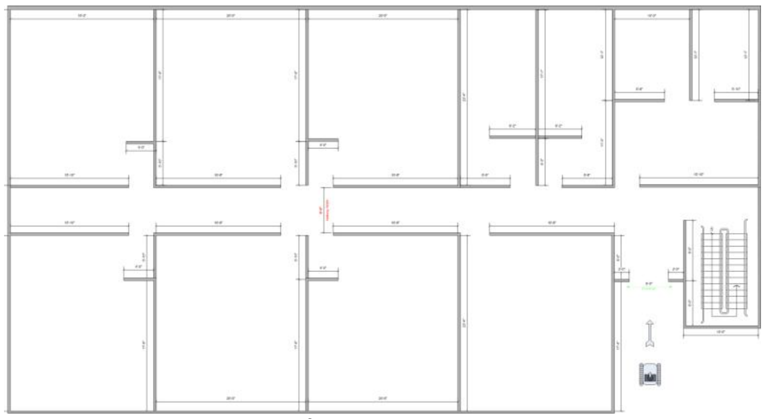


Figure 6. Ground truth map of the school house the robot explored during the FY08 performance tests.

3.1.2 Exploration Analysis Methodology

Exploration is a complex behavior for which performance was measured in a variety of ways. Our desired characteristics of robotic exploration are reduced operator workload (increased robot autonomy), rapid time to complete, and quality and accuracy of the data characterizing the environment. Many techniques for optimizing the performance of one of the metrics can have significant adverse affects on the others. For example, increasing the speed of exploration by not exhaustively searching every possible SLAM map frontier increases the likelihood of producing an incomplete map.

The configuration of ACS at the FY08 Camp Pendleton tests was designed to provide a reasonable balance between autonomy, speed, and accuracy. However the initial ACS exploration testing placed a priority on accuracy. The robot was speed-limited to a velocity significantly lower than the capability of either the hardware or software, and operator intervention was limited to situations where no intervention would cause risk to the robot or mission failure.

The results reported here are from three complete exploration "missions" at Camp Pendleton in FY08, where a mission is composed of the following sequence:

1. The robot's starting position is outdoors, adjacent to the building to be explored.
2. Using MOCU, the user designates a boundary around the building to be explored with a line-drawing function on aerial imagery. This bounds the exploration task.
3. The robot is directed into the building with geodetic waypoints on aerial imagery.
4. The robot is tasked to explore the building, at which point the robot begins fully autonomous operation.
5. When the user recognizes a state of complete exploration (all areas of the building have been completely mapped), the user directs the robot to perform a "retro-traverse," or a return to the starting position.
6. The mission is complete upon the robot's return to the starting position.

We report the performance results of the exploration mission in three classes of metrics: exploration speed, operator workload, and mapping accuracy.

3.1.2.1 Exploration Speed

Exploration speed is a measure of the time required to complete a mission. There are many factors which influence speed, including varying strategies for selecting goals and planning paths. We report just the raw numbers here (table 1). It would be desirable to be able to compare the numbers to an optimal speed, or the speed with which a human could perform the same test, but such testing was outside the scope of our initial effort. The details of the size of the areas explored are reported in section 3.1.3.1.

Run #	Time (min)	Distance (m)
1	25:23	356.36
2	30:41	297.99
3	28:41	313.07
Averages	28:09	322.5

Table 1. Actual time to execute and associated distance

3.1.2.2 Operator Interventions

We do not attempt to characterize true operator workload in these results; instead, we count the number of times during a test run that the operator had to intervene with the operation of the robot (table 2). Interventions fell into two categories: 1) extraction of the robot from a situation where it is "stuck", and 2) removal or skipping of an explore goal.

Run #	# of Interventions
1	8
2	5
3	4

Table 2. Number of operator interventions for each exploration test run

The first category usually occurred when the robot was navigating in an extremely cluttered area and entered a situation where the autonomous “Get Unstuck” behavior failed to effectively navigate to a location where the robot could effectively continue the mission. When the robot was stuck in a continuous loop or stopped completely, the user supplied waypoints or direct control to move the robot to a better position. This type of intervention generally took about 5-15 seconds of operator time.

The second category occurred in situations where the robot was trying to reach an exploration goal point that was unreachable, or the user identified as extremely inefficient. In these cases, the user chose to skip or delete the goal point with the press of a button. This type of intervention took less than a second to perform.

3.1.3 Mapping Analysis Methodology

Map accuracy of the SLAM maps were judged by two criteria: topological accuracy and dimensional accuracy. Topological accuracy is defined as the correct identification of individual rooms in the building and is reported as the detection rate of rooms versus ground truth. The false positive rate is subtracted from the detection rate to give a single accuracy indicator. Dimensional accuracy is the comparison of room and building dimensions versus ground truth. Because all the rooms in the test environment were purely rectangular to the accuracy of the hand measuring tools, only length and width were used. Accuracy is reported as the percent difference between measured and calculated length and width for each room, and for the whole building.

Because the output of ACS is simply a two dimensional occupancy grid of everything in the environment, it can be somewhat difficult to compare to a hand-measured reference containing only the dimensions of the walls. For example, bookcases or clutter up against the wall are merged with the wall in the grid, making it impossible to distinguish between the two in an occupancy grid. It is often difficult to identify the precise pixel boundaries of either corners or walls.

To help counter this problem, an automated image processing system was developed at SSC Pacific to detect each room in the building. This approach removes any subjective element in the identification of rooms or dimensions but also possibly introduces measurement errors given the noisy nature of the imagery and the statistical algorithms used in the image processing. This processing error, however, is likely very small and is included in the overall results to avoid misrepresenting the accuracy of the maps.

The image processing operates by making assumptions about interior environments. Althought these assumptions are not valid for all buildings, they were deemed valid for the buildings used in our testing:

- Buildings and rooms are constructed of rectangular shapes with flat planar walls
- Buildings contain rooms and hallways
- Rooms have aspect ratios of 3 or smaller, and everything else is a hallway
- Doorways are gaps in walls of between 0.75 and 1.75 meters
- Rooms have a size of at least 9 square meters

The image processing steps work in the following sequence:

- Find a contour around each free-standing “island” in the occupancy grid
- Remove clutter (contours encompassing less than 9 square meters of area)
- Perform image morphology which emphasizes rectangular regions, but retains their size

- Perform a second contour fit
- Run a polygon fitting algorithm on each contour
- Reject contours which do not meet the assumptions listed above

The remaining polygons are then classified as buildings, rooms, or hallways according to size and aspect ratio. Anything falsely classified at this stage is counted as a false alarm, and anything missed is a missed detection. The length and width of each building, room, and hallway is calculated as the length and width of the bounding rectangle around each approximated polygon. In the future, non-rectangular rooms can be accounted for by altering this process.

3.1.3.1 Dimensional Accuracy Results

Table 3 shows the hand-measured dimensions of each room, along with the SLAM-calculated dimensions for each of three runs. The final two columns are the mean error for each room's width and height. It is apparent that the errors are quite small except in rooms 8, 10, and 13, and the causes of the errors in those rooms will be discussed in section 3.1.3.2.

Room #	Actual Width (m)	Actual Length (m)	Run #1		Run #2		Run #3		Mean% Error Width	Mean% Error Length
			SLAM Width (m)	SLAM Length (m)	SLAM Width (m)	SLAM Length (m)	SLAM Width (m)	SLAM Length (m)		
1	7.21	5.84	7.20	5.91	6.80	5.91	7.10	5.91	0.00	0.01
2	7.10	5.84	7.10	5.91	7.26	5.81	7.20	5.91	0.01	0.01
3	7.10	6.10	7.00	6.13	6.90	6.13	7.10	6.13	0.00	0.00
4	7.10	6.10	7.20	6.13	7.10	6.23	7.20	6.23	0.01	0.02
5	7.10	6.10	7.10	6.13	7.10	6.13	7.20	6.03	0.01	0.01
6	7.10	6.10	7.20	6.13	7.10	6.13	7.30	6.13	0.03	0.00
7	7.10	6.10	7.20	6.33	7.20	6.13	7.20	6.13	0.10	0.00
8	1.90	2.84	1.53	2.90	1.53	2.70	1.53	2.80	0.19	0.01
9	5.20	2.84	5.16	2.90	5.16	2.80	5.06	2.90	0.03	0.02
10	1.90	2.84	1.63	2.80	1.43	2.70	1.53	2.80	0.19	0.10
11	5.20	2.84	4.56	2.90	4.46	2.90	0.36	0.20	0.00	0.00
12	3.42	6.15	3.14	6.13	3.04	6.13	3.14	6.13	0.08	0.00
13	3.68	3.05	3.76	3.01	3.76	3.11	3.36	2.21	0.09	0.27
14	3.68	3.10	3.66	2.72	3.66	2.72	2.96	2.72	0.00	0.12
Entire Building	16.05	29.87	16.18	31.28	17.18	30.01	16.48	30.21	0.03	0.01

Table 3. Mapping dimension accuracy results. (All dimensions are in meters and are reported with centimeter precision).

3.1.3.2 Topological accuracy results

Table 4 shows the number of false positives and detection rate for each run. Note that the detection rate is subtracted from the false positive rate in order to give a single accuracy indicator. Details of each run, including the causes for the false detections are described below.

	Run #1	Run #2	Run #3
# False Pos.	1	1	0
Detection Rate	94%	87%	94%

Table 4. Number of false positives and detection rate for each exploration test run.

<u>Run #1</u>

ess

Figure 7 shows the input and output data used to calculated topological correctness for run #1. The primary error in run #1 was the detection of two rooms identified as rooms 6 & 7 where only one room actually exists. This was caused by a couch, noticeable in the input image, which caused a misclassification as a door. The only solution to this issue is likely the use of 3D or visual data to assist in object classification. It should be noted that room #2 is really a stairwell. This was not treated as a misclassification since the perceptions available to the robot were not sufficient to detect stairs.

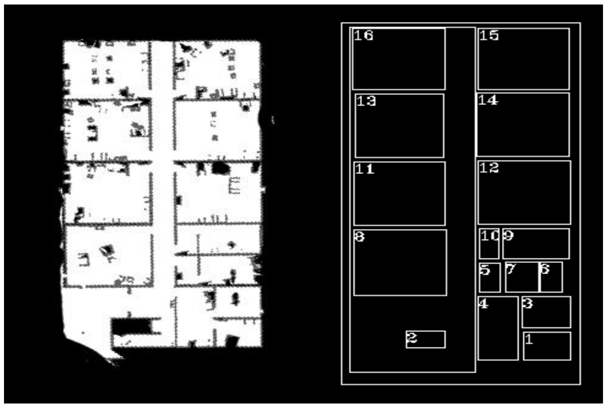


Figure 7. Run #1 raw SLAM result (left) and the calculated topology map (right)

<u>Run #2</u>

This run suffered from the same problem as run #1 and had an additional error: a missed detection of the room to the left of room #6. This missed classification was likely due to the combination of noise in the processing algorithm, and the fact that the room was so small that its size put it at the threshold of what was defined as a room. If this threshold were lower, however, the probability of false positives would increase, and so there is still considerable research to be done on what should characterize a room. Also note that clutter on the left wall of room 15 likely resulted in an underestimate of the width of that room. Such small inaccuracies due to clutter are apparent in several cases in all three runs and likely account for most measurement errors that exceed 1%.

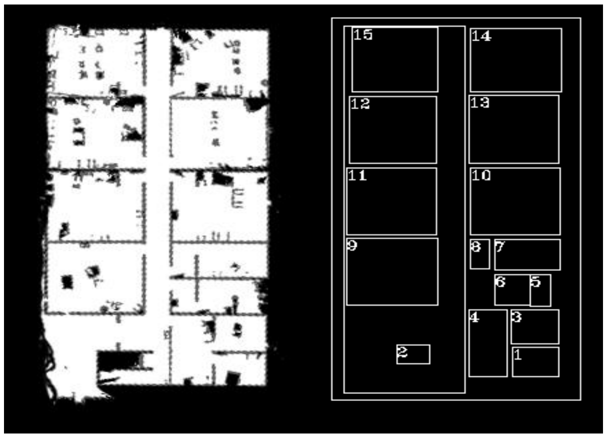


Figure 8. Run #2 raw SLAM result (left) and the calculated topology map (right)

<u>Run # 3</u>

The error in run #3 is obvious: missed detections of the two rooms that caused all the problems in the previous two maps. However this time, the fault is not with SLAM, but the premature termination of the exploration behavior by the operator.

2.1 Adaptive Localization

To test the performance of the ACS adaptive localization method, the robot was always commanded to return to its starting position in order to compare its actual start and end location; this gave the position error over the entire distance traveled. Because the UrbEE objective is to test in realistic environments as much as possible, we have tested in indoor, outdoor, off-road, and open terrain environments during single test runs. The results presented here, however, are from those from FY08 at Camp Pendleton that were focused on the building exploration mission, and thus the majority of the distance traveled was indoors and on paved road.

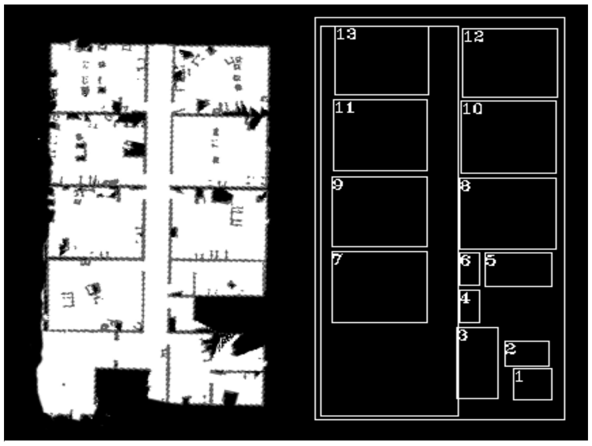


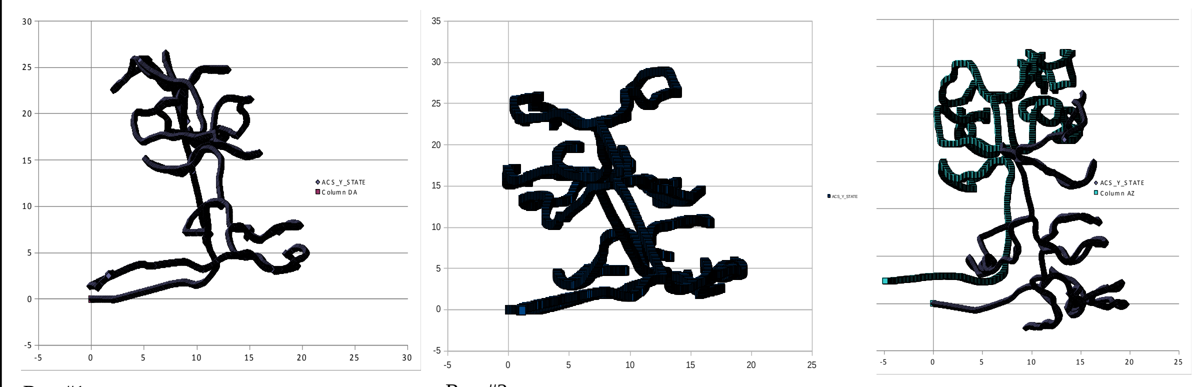
Figure 9. Run #3 raw SLAM result (left) and the calculated topology map (right)

2.2.1 Test Results

Analyzing the data from completed runs in FY08 shows that we are able to achieve an average error of < 0.8% over distance traveled, and always achieved an error of < 1.2% over distance traveled. Notice that the AEKF did not use any input from SLAM, which would have greatly increased the accuracy of the state estimates. Some of the data was discarded due to runs that were never completed because the mapping behavior failed. From the completed test runs that produced errors above our desired value of 1.0%, we were able to discover and fix errors in the configuration options of the AEKF or in the AEKF algorithm itself. This led to better results as testing advanced; we will be decreasing our desired error value to 0.65% for the FY09 tests. Table 5 summarizes the performance of six completed runs; figures 10 and 11 show their corresponding X/Y_STATE graphs.

Run	Time (sec)	Distance Traveled (m)	Error X-Y-Z (m)	Error as % of Distance Traveled
1	1756	261.78	3.01	1.15
2	1700	356.36	1.26	0.35
3	1611	443.49	5.47	1.23
4	1727	313.07	1.96	0.63
5	1815	297.99	2.52	0.85
6	1903	162.8	0.96	0.59

Table 5. Performance of ACS AEKF during six completed localization test runs.

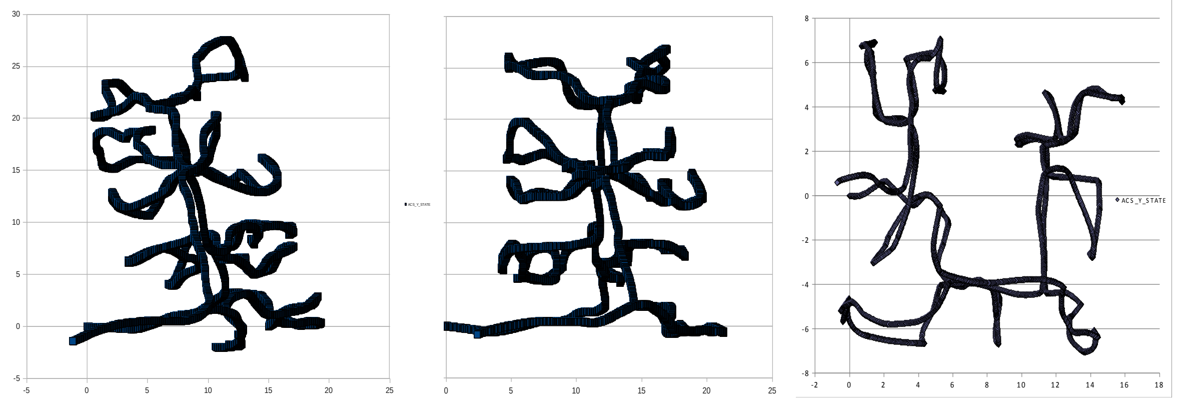


Run #1:
Distance traveled in school house: 261.78 m
Error X-Y-Z: 3.01 m
%Error of distance traveled: 1.15 %

Run #2:
Distance traveled in school house: 356.36 m
Error X-Y-Z: 1.26 m
%Error over distance traveled: 0.35%

Run #3:
Distance traveled in school house: 443.49 m
Error X-Y-Z: 5.47 m
%Error over distance traveled: 1.23%

Figure 10. X/Y_STATE graphs of localization test runs #1 – 3.



Run #4:
Distance traveled in school house: 313.07 m
Error X-Y-Z: 1.96 m
%Error of distance traveled: 0.63%

Run #5:
Distance traveled in school house: 297.99 m
Error X-Y-Z: 2.52 m
%Error of distance traveled: 0.85%

Run #6:
Distance traveled in city hall: 162.8 m
Error X-Y-Z: 0.96 m
%Error of distance traveled: 0.59%

Figure 11. X/Y_STATE graphs of localization test runs #4 – 6.

IV. CONCLUSION

The UrbEE project will continue through FY11, which will include continued performance testing at Camp Pendleton and other MOUT sites. Our objective is to increase the difficulty of the environmental variables to fully evaluate the realistic use of semi-autonomous ground robots to support dismount troops operating in urban areas.

As mentioned throughout the paper, FY08 testing was primarily focused on exploration of single-story building structures. In FY09, our objectives will be to explore multiple, single-story buildings within at least a .5m radius. We expect that the behaviors described in this paper will require further expanded capabilities, especially to accommodate exploration of larger areas. An example of planned new mapping functions include the ability to save, load, and restore both completed and uncompleted maps so that an operator can end an exploration mission at any time without losing already produced map data, and a robot can pick-up exploring an area where it (or another robot) left off. The need to traverse outdoors increases as the robots will move from building to building; therefore, an ACS module will be developed to detect negative obstacles and perceive/negotiate uneven terrain.

In FY10, the exploration area and number of buildings will increase, including an added major challenge to explore multi-story buildings. Performance tests on the detection, climbing, and descending of varying staircase configurations will be included in our test scenarios, as well as the mapping and localization during exploration of multiple floors of the same building. Partially damaged buildings will also be introduced, thus further testing the capabilities of our negative obstacle detection. Usability tests will begin in FY10 with robotic users in order to validate the performance and focus on optimization of any subset of behaviors in FY11.

In FY11, we will focus on both performance and usability testing throughout the year. The exploration area and number of buildings, including multiple multi-story buildings and partially damaged buildings, will increase. Performance tests will focus on the entire robotic system as a whole and will be conducted at as many test sites as possible to obtain as large of a data set as we can get against a wide range of urban environment variables possible.

SSC Pacific's goal is to provide the warfighter with efficient and reliable robotic capabilities. While the improvements made to the behaviors described here were done at SSC Pacific, some were originally developed and done in collaboration with other government agencies, academia, and private industry. SSC Pacific is eager to partner with any R&D efforts with similar goals to enhance robotic capabilities and functionalities supporting urban operations to reduce the warfighter's workload and increase their situational awareness, mission success, and overall chance of survival.

REFERENCES

1. Powell, D., Gilbreath, G., and M. Bruch, "Multi-Robot Operator Control Unit for Unmanned Systems", *Defense Tech Briefs*, August 1, 2008

2. Sights, B., Ahuja, G., Kogut, G., Pacis, E.B., Everett, H.R., Fellars, D., and S. Hardjadinata, "Modular Robotic Intelligence System Based on Fuzzy Reasoning and State Machine Sequencing", SPIE Proc. 6561: Unmanned Systems Technology IX, Defense & Security Symposium, Orlando, FL, April 9-13, 2007.

3. www.kartorobotics.com

4. Pacis, E.B, B. Sights, G. Ahuja, G. Kogut, H.R. Everett, "An adapting localization system for outdoor/indoor navigation," SPIE Proc. 6230: Unmanned Systems Technology VIII, Defense Security Symposium, Orlando FL, 17-20 April 2006.